Computation of Initial State
for Tail-Biting Trellis

A thesis presented to
the faculty of the
Fritz J. and Dolores H. Russ
College of Engineering and Technology of
Ohio University

In partial fulfillment
of the requirements for the degree
Master of Science

Yiqi Chen
June 2005
This thesis entitled
COMPUTATION OF INITIAL STATE
FOR TAIL-BITING TRELLIS

By
Yiqi Chen

has been approved for
the School of Electrical Engineering and Computer Science
and the Russ College of Engineering and Technology by

Jeffrey C. Dill
Professor of School of Electrical Engineering and Computer Science

Dennis Irwin
Dean of Fritz J. and Dolores H. Russ
College of Engineering and Technology
ABSTRACT


Computation of Initial State for Tail-Biting Trellis (63 pp.)

Director of Thesis: Jeffrey C. Dill

Prior to this thesis, a state assignment algorithm for a family of tail-biting trellis codes was presented (Lo, 1997; Lopez-Permouth, Dill and Lindsey, 2004). Based on this premise, this thesis develops and clarifies the algorithm for calculation of the initial states to a CTCM trellis by employing a source alphabet size of four. State mapping is necessary to be done before using the formula to calculate the initial states. In addition, the algorithm is implemented in computer programs of Matlab® and C++ that perform the calculation of the initial state for any given information sequence. Experimental results and analysis are provided for the performance comparisons. It is shown that various changes in CTCM parameters lead to expected corresponding changes in the bit error performance.

Approved:

Jeffrey C. Dill
Professor of School of Electrical Engineering and Computer Science
ACKNOWLEDGEMENTS

I would like to express my deepest thanks to my advisor, Dr. Jeffrey C. Dill, for his guidance and instruction in directing this research. There is no doubt that without his guidance, all this would not have come true.

I would also like to thank all the members of my thesis committee, Dr. David Matolak, Dr. Sergio Lopez-Permouth, and Dr. Joseph Essman, for their time and effort. I especially thank Dr. David Matolak for his revising of my thesis.

Lastly, I thank my parents and friends for their love and support during my entire graduate career.
Table of Contents

ABSTRACT........................................................................................................................3
ACKNOWLEDGEMENTS.........................................................................................................4
LIST OF TABLES....................................................................................................................7
LIST OF FIGURES..................................................................................................................8
CHAPTER 1 INTRODUCTION...............................................................................................9
  1.1 History........................................................................................................................9
  1.2 Contributions of the Thesis......................................................................................11
  1.3 Outline of the Thesis...............................................................................................12
CHAPTER 2 BACKGROUND...............................................................................................14
  2.1 Error Correcting Codes and Digital Communication Systems.............................14
    2.1.1 Digital Communication Systems......................................................................14
    2.1.2 Error Correcting Codes..................................................................................15
    2.1.3 Trellis Codes...................................................................................................17
  2.2 Trellis Termination..................................................................................................20
    2.2.1 Zero Tailing and Tail Biting............................................................................21
    2.2.2 Circular Trellis Coding with Permuted State Structure..................................22
  2.3 Trellis-Coded Modulation......................................................................................23
    2.3.1 Conventional Trellis-Coded Modulation (TCM)..............................................23
    2.3.2 Circular Trellis-Coded Modulation (CTCM)...................................................24
  2.4 Spread Spectrum Communications.......................................................................25
CHAPTER 3 DESIGN OF STATE TABLE........................................................................27
  3.1 Zech’s Logarithm...................................................................................................27
    3.1.1 Background: Field and Galois Field...............................................................27
    3.1.2 Representation of Galois Field.......................................................................28
    3.1.3 Lemmas of Galois Field..................................................................................30
    3.1.4 Zech’s Logarithm.........................................................................................31
    3.1.5 Calculation Based on Zech’s Logarithm..........................................................33
  3.2 Permuted State Transition Table............................................................................35
    3.2.1 A Good Family and State Permutation..........................................................35
    3.2.2 State Table....................................................................................................39
  3.3 Initial States.............................................................................................................41
    3.3.1 Theorem and Proof of Calculation of Initial State..........................................41
    3.3.2 Mapping of Information Sequence for the Calculation of Initial States............43
    3.3.3 Example of Calculation of Initial State............................................................43
CHAPTER 4 PERFORMANCE............................................................................................47
  4.1 Comparison of Bit Error Probabilities of CTCM Using Old and New Programs....47
  4.2 Comparison of Bit Error Probabilities of CTCM with Different Block Sizes......48
  4.3 Comparison of Bit Error Probabilities of CTCM with Different Dimensions.....50
  4.4 Comparison of Bit Error Probabilities of CTCM with Different Trellis Depths....51
CHAPTER 5 CONCLUSIONS.............................................................................................53
  5.1 Summary..................................................................................................................53
  5.2 Future Research.....................................................................................................54
REFERENCES..................................................................................................................55
APPENDIX A ................................................................................................................... 57
Zech’s Log Table for GF(2^6), Defined by \( f(x) = x^6 + x + 1 \) ........................................ 57
APPENDIX B ................................................................................................................... 58
Permutated State Transition Table for 64 States ............................................................ 58
APPENDIX C ................................................................................................................... 60
Matlab® Program of Calculation of Initial States ......................................................... 60
APPENDIX D ................................................................................................................... 62
C++ Program of Calculation of Initial States ................................................................. 62
LIST OF TABLES

Table 3.1  Exponential and polynomial representation of Galois field elements .............. 29
Table 3.2  Representation of Galois field elements in vector space ................................. 30
Table 3.3  Zech’s log table for GF(2^3), defined by $f(x) = x^3 + x + 1$ ............................... 32
Table 3.4  Zech’s log table for GF(2^4), defined by $f(x) = x^4 + x + 1$ .............................. 33
Table 3.5  A good family as $\sigma_i$ style for sixteen states.................................................... 37
Table 3.6  A good family as state permutation style for sixteen states................................. 38
Table 3.7  State transition table for 16 states .................................................................... 40
Table 3.8  State mapping from the paper to the computer program ...................................... 46
LIST OF FIGURES

Figure 2.1  The structure of digital communication system ............................................. 14
Figure 2.2  General structure of the channel encoder with block codes .......................... 16
Figure 2.3  (2,1,2) convolutional encoder (code rate = 1/2, K=2) ............................... 17
Figure 2.4  State diagram for the (2,1,2) encoder in Figure 2.3 ................................. 18
Figure 2.5  Trellis diagram for the (2,1,2) encoder in Figure 2.3 ............................... 19
Figure 2.6  Increase of signal set size for trellis-coded modulation .............................. 23
Figure 3.1  Trellis diagram of transmission-state set \{\sigma_{x}, \sigma_{12}, \sigma_{14}, \sigma_{5}\} .......... 41
Figure 4.1  Error probabilities comparison between old and new programs ................. 47
Figure 4.2  Error probabilities for B = 24, 48 and 96, and BPSK ............................... 49
Figure 4.3  Error probabilities for N = 12, 24 (S = 64) and BPSK .............................. 50
Figure 4.4  Error probabilities for S = 16, 64 and BPSK .......................................... 51
Chapter 1 will present some history of the research on error correcting coding and trellis-coded modulation. The primary contribution of this thesis research will also be provided. The chapter will conclude with an outline of the thesis.

1.1 History

Error correcting coding can be defined as the introduction of redundancy to a message in order to detect or correct errors which occur during the message transmission over a noisy channel. There are two primary types of error correcting codes: block codes and convolutional codes. Both types of codes provide the ability of error-detection or error-correction by adding redundant data to the original source data. The main difference between block codes and convolutional codes is that in block codes, source data is divided into finite blocks, and encoder and decoder operate on each block independently, whereas convolutional codes need not (although in practice they always do). Both block codes and convolutional codes require additional transmission bandwidth when transforming each input data k-tuple into a larger output codeword n-tuple. They provide improvements in error performance at the cost of bandwidth expansion. Therefore, in the past, coding generally was not popular for band limited channels, where signal bandwidth expansion was not practical.

In 1982, Ungerboeck proposed the trellis-coded modulation (TCM) which combines modulation and coding schemes (Ungerboeck, 1982; Ungerboeck, 1987). TCM has been extensively researched and has achieved remarkable commercial success in its
application. It combines a state-oriented coding scheme (such as convolutional codes) and a multi-level/phase modulation into one single step for preventing bandwidth expansion. Coding gain is achieved without sacrificing data rate or without increasing either bandwidth or power.

Recently, circular trellis-coded modulation (CTCM) with the permuted state structure for power-limited spread spectrum channels has been investigated (Lo, 1997; Alder, 1998; Song, 2000). CTCM takes the basic concepts of TCM and applies them to achieve coding gain on a power-limited spread spectrum channel.

In trellis coding, there is an important technique called trellis termination, which forces the encoded trellis path to satisfy the state constraint – the starting state and ending state should be the same (Sklar, 1988). In conventional TCM, which is typically a shift-register based trellis coding environment, the initial state of the encoder is always assumed to be state 0, and zero tailing is used for trellis termination (Sklar, 1988; Ma and Wolf, 1986). In this method, a certain number of zeros are added to the end of the input sequence to force the encoded trellis path to end at state 0. However, the code rate is reduced due to these added zeros. This can be significant when the input information sequence has a short length.

An alternative trellis termination method without code rate loss is called tail biting, which was first introduced by Solomon and Van Tilborg and generalized by Howard H. Ma et al (Ma and Wolf, 1986). The state constraint is satisfied by forcing every trellis path to be a circular path -- a path having the same starting state and ending state, which can be any of the possible states in the trellis.
However, for a long time, zero tailing has been the only practically used trellis termination method because of its simplicity and the fact that many efficient decoding algorithms were invented based on that condition.

With the recent development of turbo codes, trellis termination began to be addressed again. A novel circular trellis coding with a permuted state structure was invented, which is a revolutionary aspect of CTCM. A permuted state transition table is built to guarantee the state constraint. Any input data block is uniquely mapped to a circular trellis path under this state transition table. This is done without code rate loss or initializing the encoder.

1.2 Contributions of the Thesis

A state assignment algorithm for a family of tail-biting trellis codes is presented in “On a Trellis-Coded Modulation Design with a Strong Tail-Biting Property” by S. Lopez-Permout and J. Dill (Lopez-Permout, Dill and Lindsey, 2004). Based on the premise of this paper, this thesis develops and clarifies the algorithm for calculation of the initial state to a CTCM trellis by employing a source alphabet size of four.

In addition, the algorithm is implemented in computer programs of Matlab® and C++ that performs the calculation of the initial state for any given information sequence.

Experimental results and analysis are provided for the performance comparisons. It is to verify the changes in the bit error performance when various changes in CTCM parameters are simulated. These results are expected analytically.
1.3 Outline of the Thesis

This thesis is organized as follows:

Chapter 2  A general communication background is presented. First, digital communication systems and error correcting codes are described for an understanding of the theory of trellis-coding modulation. Trellis termination is then explained, which is an important property of trellis-coded modulation. Then Trellis-coded modulation is discussed, which contains conventional trellis-coded modulation and circular trellis coding with permuted state structure. The latter version is the main focus of this thesis. In the end of Chapter 2, a brief introduction of spread spectrum communication is reviewed.

Chapter 3  This chapter focuses on the design of the state table and the calculation of the initial state. Zech’s logarithm is introduced first, which makes the required calculations in the Galois field much easier. Through the Zech’s logarithm, a good family of state permutations can be constructed. After four elements are picked from the good family, the state table is built. The state table indicates what the next state is for any current state, driven by all possible transmission signals. Given any data sequence, in order to obtain a circular trellis path, one and only one initial state exists. A formula to calculate the initial state is given and proven in the end, and an example is also shown.

Chapter 4  The bit error probability performance of CTCM is investigated. Comparisons are drawn for showing the affects of different CTCM parameters.

Chapter 5  A summary of the results of this thesis is presented, and further research is discussed.
Appendices  Zech’s log table and permutated state transition table for 64 states are listed. Computer programs, in the form of MATLAB source and C source files, are included for the details of developed algorithms.
CHAPTER 2  BACKGROUND

A general communication background will be presented in this chapter. First, digital communication systems and error correcting codes will be described to aid in understanding of the theory of trellis-coding modulation. Trellis-coded modulation will also be discussed, as well as the important property of trellis termination. In the end of this chapter, the brief introduction of spread spectrum communication will be reviewed.

2.1 Error Correcting Codes and Digital Communication Systems

2.1.1 Digital Communication Systems

Digital communication systems have a well-defined structure and knowledge of this structure is helpful in understanding the role of coding. The simplest structure, shown in Figure 2.1, is for a point-to-point communication system (Biglieri, et al., 1991).

![Figure 2.1 The structure of digital communication system](image-url)
Binary words are taken in from the information source, which means that the elements in the information sequence are only 1s and 0s. The source is followed by a source encoder. The role of the source encoder is to remove redundancy (in effect randomizing the source data), which makes efficient utilization of channel -- an often scarce resource of telecommunications. The goal of the channel encoder is to introduce an error correction capability into the source encoder output to combat channel transmission errors. In this thesis, trellis coding is used in the channel encoder for adding redundancy to the source outputs for the receiver to correct errors occurred during signal transmission. The modulator interfaces the channel encoder to the channel. Trellis coding serves to merge the processes of modulation and coding. Each block after the noisy channel is the inverse of a corresponding block before the channel.

2.1.2 Error Correcting Codes

In any communication systems, some errors can occur during the signal transmission because of the noisy channel. Therefore, an error correcting code is often applied. An error correcting code is also called an error control code, which introduces a controlled redundancy to the information sequence to provide a way for the receiver to correct the errors. The error correcting code is applied in the channel encoder.

2.1.2.1 Block codes

Error correcting codes can be divided into two general categories, block codes and convolutional codes. In block codes, source data is divided into blocks of fixed length denoted by $k$. The encoder and decoder operate on each block independently. A block
code normally is denoted as $(n,k)$, which means each $k$-symbol input block transforms to an $n$-symbol output data block in the encoder $(n>k)$. The ratio $k/n$ is defined as code rate. The redundant $(n-k)$ bits are used to correct errors at the decoder. Figure 2.2 is the general structure of the channel encoder with block codes.

![Figure 2.2 General structure of the channel encoder with block codes](image)

Code rate: $k/n$

2.1.2.2 Convolutional codes

Convolutional codes were first introduced by Elias in 1955 (Wilson, 1996). In contrast to block codes, convolutional codes need not operate on independent blocks of source data. The convolutional code is described by three parameters, $n$, $k$ and $K$, denoted as $(n,k,K)$. The $n$-tuple encoder output at any given time depends not only on the $k$-tuple input at that time, but also on the previous $K$ input $k$-tuples. $K$ is known as the constraint length, which defines the number of $k$-tuple stages in the shift register of the encoder. $k/n$ is the code rate, the same as for block codes. It is obvious that a convolutional encoder has memory. This is also an important difference between convolutional codes and block codes, which are memoryless between blocks.
Convolutional codes play a central role in this thesis as they are commonly used to generate trellis codes.

An example convolutional encoder with one input source bit ($k=1$) and two output bits ($n=2$) is given in Figure 2.3. This code has rate $\frac{1}{2}$. The number of memory elements in the shift register is two, so this called a (2,1,2) encoder.

![Figure 2.3 (2,1,2) convolutional encoder (code rate = 1/2, K=2)](image)

2.1.3 Trellis Codes

Convolutional code is known as the most popular trellis code. The trellis encoder is a finite-state machine (FSM), which is the general name given to a device that has a memory of past signals, and finite refers to the fact that there are only a finite number of unique states that the machine can encounter. That is, the trellis encoder output at a given time depends on the input at that time, as well as the current state of the encoder. There
are two representations which are frequently used to describe the encoder, the state
diagram and trellis diagram [Sklar, 1988].

2.1.3.1 State Diagram Representation

The state diagram illustrates the state transitions related to each possible $k$-tuple input. Figure 2.4 is the state diagram for the encoder in Figure 2.3. The ovals denote the states in the shift register. The state variables in this example are 00, 10, 01, 11, and represent the contents of the memory elements in the shift register of Figure 2.3. The branches are labeled by the possible $k$-tuple input string in a given time slashed with the encoder output at that time.

![State Diagram for the (2,1,2) Encoder in Figure 2.3](image)

Figure 2.4 State diagram for the (2,1,2) encoder in Figure 2.3
2.1.3.2 Trellis Diagram Representation

From the state diagram, a trellis diagram can be derived, which extends the state diagram along the time dimension and merges the same state at the same time instance. This representation can be a more clear and useful method for analysis. Figure 2.5 is the trellis diagram for the encoder in Figure 2.3.

The nodes represent trellis states, and the branches represent state transitions. The branches are labeled by the output of the encoder when the state transition occurs. The trellis code is started at time $t_0$, and increases by one at each time instance. All states can be reached by a certain number of trellis levels, and then a fixed trellis “section” prevails. The trellis section completely describes the code.

Figure 2.5  Trellis diagram for the (2,1,2) encoder in Figure 2.3
In this thesis, the circular trellis code is investigated. A circular trellis encoder refers to a type of trellis encoder, in which each information sequence is mapped to a circular trellis path—a path having the same starting state and ending state. This state can be any of the states in this trellis and is completely determined by the information sequence itself. Thus, since each information sequence is associated with a unique starting/ending state, this state must be precomputed before transmission, unlike the case in the typical zero-tailing approach.

All the information contained by the state diagram and trellis diagram can be recoded in a table which is called state transition table. This table indicates what the next state is for any state driven by all possible $k$-tuple inputs. Building a trellis code can be reduced to specifying its state transition table, which will be explained in next chapter in detail.

### 2.2 Trellis Termination

In trellis coding, the information sequence is separated into several blocks with fixed length $L$ to have a block structure. The value of $L$ is determined a priori. The trellis encoder operates on each block independently. In order to provide the same error protection to the last information symbol, a technique called trellis termination is employed. Trellis termination is an important characteristic in trellis coding, which forces the trellis path to satisfy the state constraint—having the same starting state and ending state. Zero tailing, tail biting, and circular trellis with permuted state structure are the three methods to achieve trellis termination. They are introduced in the following two subsections.
2.2.1 Zero Tailing and Tail Biting

In conventional TCM, which is typically a shift-register based trellis coding environment, the decoder must know the starting state of the encoder before transmission. Therefore, the initial state of the encoder is always assumed to be state 0, which means the contents of the shift register are all zeros initially, and zero tailing is used for trellis termination (Sklar, 1988; Ma and Wolf, 1986). In this method, a certain number of zeros are added to the end of the terminated input sequence to force the encoded trellis path to end at state 0. Let’s consider a binary \((l, r, K)\) convolutional code. In this case, \(K\) zeros must be appended. For an information sequence of length \(L\), the resultant code is a \((L+K)r, L\) block code, with a code rate \(L/((L+K)r)\). The quantity \(K/(L+K)\) is the rate loss which is the ratio of the code rate with \(K\) appended zeros to the code rate without \(K\) appended zeros. It is obvious that the code rate is reduced due to these padded zeros. It especially becomes significant when the input information sequence has a very short length. To reduce the rate loss, the truncation period \(L\) is generally made as long as possible.

An alternative trellis termination without code rate loss is called tail biting, which was first introduced by Solomon and Van Tilborg and generalized by Howard H. Ma et al (Ma and Wolf, 1986). For a binary \((l, r, K)\) encoder, we first initialize the encoder by inputting the last \(K\) information bits of the information sequence into the encoder and ignore the output for \(Kr\) output symbol times, then input all \(L\) information bits and take the resultant \(Lr\) output bits as code word. The last \(K\) information bits will force the encoded path to go back to the initial state of the encoder. This state is totally determined by the input information sequence itself. The resulting code is a \((Lr, L)\) block code of
rate $1/r$. It is a type of circular trellis coding. The state constraint is satisfied by forcing every trellis path to be a circular path -- a path having the same starting state and ending state, which can be any of the possible states in the trellis. Compared to zero tailing, the tail biting has no code rate loss caused by adding zeros.

Zero tailing has been the only practically used trellis termination method for a long time because of its simplicity, and the fact that many efficient decoding algorithms were invented based on that condition.

### 2.2.2 Circular Trellis Coding with Permuted State Structure

With the recent development of turbo codes, trellis termination began to be addressed again. A novel circular trellis coding scheme with a permuted state structure was invented, which is a revolutionary aspect of circular trellis-coded modulation (CTCM) (Lopez-Permouh, Dill and Lindsey, 2004). This is the emphasized trellis termination method in this thesis and will be introduced in detailed in later chapters.

A permuted state transition table is built to guarantee the state constraint. Any input data block is uniquely mapped to a circular trellis path under this state transition table. The same starting state and ending state is a function of the symbols in the information sequence. This coding scheme is done without code rate loss (zero padding) or initializing the encoder.
2.3 Trellis-Coded Modulation

2.3.1 Conventional Trellis-Coded Modulation (TCM)

Trellis-coded modulation (TCM) was invented to achieve error-performance improvement without expansion of signal bandwidth. TCM combines a state-oriented coding scheme (such as convolutional code) and a multi-level/phase modulation into one single step for preventing bandwidth expansion. A larger signal constellation is used to provide the needed coding redundancy while keeping the same channel symbol rate (bandwidth) and average power as those of an uncoded system. (Biglieri, et al., 1991)

Figure 2.6a illustrates a signal space diagram for an uncoded 4-ary PAM signal set, before and after being rate 2/3 encoded into an 8-ary PAM signal set. Similarly, Figure
2.6b illustrates a signal space diagram for an uncoded 4-ary PSK signal set, before and after being rate 2/3 encoded into an 8-ary PSK signal set. In each of the cases shown in Figure 2.6, the system is configured to use the same average signal power before and after coding. The increase in the alphabet size does not result in an increase in required bandwidth.

2.3.2 Circular Trellis-Coded Modulation (CTCM)

Circular trellis-coded modulation (CTCM) with the permuted state structure is the backbone of this thesis and has been investigated (Lo, 1997; Alder, 1998; Song, 2000). It takes the basic concepts of TCM and applies them to achieve coding gain on a power-limited spread spectrum channel. CTCM can be viewed as a block code with trellis structure in the sense that source data is encoded block-by-block independently. That is, individual source data blocks are mapped into a particular permuted path through the trellis. A high dimensional signaling constellation is used so that the CTCM is intended for use in a power-limited channel, as opposed to the use of conventional TCM for a band-limited channel.

Let’s define some parameters used in a general circular trellis-coded modulation system. CTCM system is normally denoted as \((N, n, D, B)\) system. \(N\) is dimensionality of the transmitted signal space. \(n\) is the number of transmission symbols. In this thesis, a source alphabet size of \(n=4\) is emphasized. \(D\) is the trellis depth, which refers to the number of transitions needed for a given state to reach any other state in the trellis, and \(B\) is input source symbol sequence length or block length which denotes the size of the input source block as well as the number of transitions in a legal trellis path. \(B\) must be
equal to or larger than $D+1$ ($B \geq D+1$). The number of trellis states in HDTCM is determined by $S=n^D$.

### 2.4 Spread Spectrum Communications

The use of spread spectrum (SS) communication systems has become very widespread in recent years. In a spread spectrum communication system, the transmission bandwidth employed is much greater than the minimum bandwidth required to transmit the information. A system is defined to be a spread spectrum system if it fulfills the following requirements (Sklar, 1988):

1. The signal occupies a bandwidth much in excess of the minimum bandwidth necessary to send the information.
2. Spreading is accomplished by means of a spreading signal, often called a code signal (pseudo-noise, AKA PN), which is independent of the data.
3. At the receiver, dispreading (recovering the original data) is accomplished by the correlation of the received spread signal with a synchronized replica of the spreading signal used to spread the information.

There are two most prevalent types of spread spectrum communication systems, which are direct sequence (DS) and frequency hopping (FH). In a DS system, the PN code is used to shift the phase of a PSK signal pseudo-randomly, which has the effect of widening the frequency spectrum of the signal. In a FH system, the PN code is used to pseudo-randomly select the carrier frequency in a FSK system, which has the effect of shifting the spectrum of the transmitted signal over a wide range of frequencies. Both
systems can be coupled with circular trellis-coded modulation (CTCM) for transmission of data over a power-limited channel.
CHAPTER 3  
DESIGN OF STATE TABLE

As stated in Chapter 2, a trellis encoder is a finite-state machine (FSM) which can be represented by a state transmission table. This chapter focuses on the design of the state transmission table which satisfies the circular state constraint. First, the Zech’s logarithm is introduced. It makes the calculation in the Galois field much easier. Through Zech’s logarithm, a good family can be constructed. After four elements are picked up from the good family, the state table is built. The state table indicates what the next state is for any state driven by all possible transmission signals. Given any transmission sequence, in order to obtain a circular trellis path, one and only one initial state exists. A formula to calculate the initial state is given and proved in the end, and an example shows the procedure of calculation.

3.1 Zech’s Logarithm

3.1.1 Background: Field and Galois Field

A field \( F \) is a set of elements which is closed under two binary operations, which are denoted by addition “+” and multiplication “·”. Both operations are associative and commutative. Additive and multiplicative identities exist, elements have inverses with respect to both operations with the exception that 0 has no multiplicative inverse, and the multiplication distributes over the addition.

The set of integers \( \mathbb{F}_q = 0, 1, \ldots, q-1 \) is a field under modulo \( q \) addition and multiplication, as long as \( q \) is a prime number. For instance, \( \mathbb{F}_5 \) is a field but \( \mathbb{F}_4 \) is not. If
F is a finite field with q elements (q is a finite number), F usually is denoted as GF(q).

GF stands for Galois field after E. Galois, a noted French algebraist of the early 19th century. Note that q must have the form \( p^m \), where \( p \) is a prime positive integer and \( m \) is a positive integer. That is, q must be a prime number or a power of a prime number.

### 3.1.2 Representation of Galois Field

The multiplication of field elements is easy when the elements are represented as powers of a generator. However, the addition of field elements is not easy when elements are given as power of a generator. Conversely, whereas addition is easy when elements are represented as polynomials, multiplication is not. It would be convenient to use a single representation in which both operations are easy (Vanstone and Oorschot, 1989; Wicker, 1995).

Let \( \beta \) be an element in GF(q) and let 1 be the multiplicative identity. Consider the following sequence of elements,

\[
1, \beta, \beta^2, \beta^3, \beta^4, \beta^5, \ldots
\]

Since \( \beta \) is contained in GF(q), all of the successive powers of \( \beta \) must also be in GF(q) by closure under multiplication. Therefore, the sequence must begin to repeat values found earlier in the sequence at some point because the number of elements in a field is finite. Therefore, the GF(q) (q is a prime number) can be represented using 0 and (q-1) consecutive powers of a primitive element. Similarly, the element in a Galois field GF(p^m), where \( p \) is a prime number and \( m \in Z^+ \), can be determined as a power of the generator \( \beta \).
In order to construct GF(q), an irreducible polynomial in GF(q) is required. Consider
\[ f(x) = x^m + a_{m-1}x^{m-1} + \ldots + a_1x + a_0 \]
as primitive in GF(q). If \( \beta \) is a root of \( f(x) \), it must satisfy
\[ f(\beta) = \beta^m + a_{m-1}\beta^{m-1} + \ldots + a_1\beta + a_0 = 0 \]. It follows that
\[ \beta^m = -a_{m-1}\beta^{m-1} - \ldots - a_1\beta - a_0 \]. The individual powers of \( \beta \) of a degree greater than or equal to \( m \) can be re-expressed as polynomials in \( \beta \) of degree \((m-1)\) or less.

Table 3.1 shows the exponential representation and polynomial representation of Galois field element of GF\((2^3)\). \( f(x) = x^3 + x + 1 \) is primitive in GF\((2^3)\). Let \( \beta \) be a root of \( f(x) \). This implies that \( \beta^3 + \beta + 1 = 0 \), or equivalently, \( \beta^3 = \beta + 1 \). \( \beta^0 = 1 \) is defined here.

<table>
<thead>
<tr>
<th>Field element</th>
<th>Exponential representation</th>
<th>Polynomial representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \beta^0 )</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>( \beta^1 )</td>
<td>( \beta )</td>
</tr>
<tr>
<td>2</td>
<td>( \beta^2 )</td>
<td>( \beta^2 )</td>
</tr>
<tr>
<td>3</td>
<td>( \beta^3 )</td>
<td>( \beta + 1 )</td>
</tr>
<tr>
<td>4</td>
<td>( \beta^4 )</td>
<td>( \beta(\beta^3) = \beta(\beta + 1) = \beta^2 + \beta )</td>
</tr>
<tr>
<td>5</td>
<td>( \beta^5 )</td>
<td>( \beta^5 + \beta^2 = \beta^2 + \beta + 1 )</td>
</tr>
<tr>
<td>6</td>
<td>( \beta^6 )</td>
<td>( \beta^6 + \beta + \beta^2 = \beta^2 + 1 )</td>
</tr>
<tr>
<td>7</td>
<td>( \beta^7 = 1 = \beta^0 )</td>
<td>( \beta^5 + \beta = 1 )</td>
</tr>
</tbody>
</table>

It is convenient to write \((a_0a_1a_2)\) for the field element \(a_0 + a_1x + a_2x^2\). Note that we have adopted the convention of ordering coefficients left side smallest (from low order to
high order). Using this notation, the elements of GF(q) can thus be interpreted as a vector space. Table 3.2 is an example of vector representation for elements in GF(2^3).

Table 3.2 Representation of Galois field elements in vector space

| $β^0$ = (100) | $β^4$ = $β^3 + β = (011)$ |
| $β^1$ = (010) | $β^5$ = $β^2 + 1 = (111)$ |
| $β^2$ = (001) | $β^6$ = $β^2 + 1 = (101)$ |
| $β^3$ = $β + 1 = (110)$ | $β^7$ = $β^0 = (100)$ |

3.1.3 Lemmas of Galois Field

The product of $β^a$ and $β^b$ is $β^a β^b = β^{a+b (mod(q-1))}$. For the GF($p^m$) with generator $β$, where $p$ is a prime number and $m ∈ Z^+$, to facilitate the addition of field elements represented as powers of a generator, a table is set up, called Zech’s log table. For each integer $i$, $0 ≤ i ≤ p^m - 2$, we determine and tabulate the integer $j = z(i)$ such that $1 + β^i = β^{z(i)}$. Then,

$β^a + β^b = β^a (1 + β^{b-a (mod p^m - 1)}) = β^a β^{z(b-a)} = β^a + z(b-a)$

and $z(b-a)$ can be obtained from the Zech’s log table.

The following lemmas are often used in dealing with finite field.

**Lemma 1** Let $i ∈ \{∞, 0, 1, ..., S - 2\}$ and $z(●)$ represent a Zech’s logarithm transformation, then

$z(z(i)) = z^2(i) = i$

Proof: $β^i = β^i + 1 + 1 = (β^i + 1) + 1 = β^{z(i)} + 1 = β^{z(z(i))}$
Lemma 2  Let \( i \in \{0,1,...,S-2\} \) and \( z(\bullet) \) represent a Zech’s logarithm transformation, then
\[
z(i) = j \iff z(j) = i
\]
Proof: \( \beta^{z(i)} = \beta^j \iff 1 + \beta^i = \beta^j \iff \beta^i = \beta^j + 1 \iff \beta^i = \beta^z(j) \)

Lemma 3  Let \( i \in \{0,1,...,S-2\} \) and \( z(\bullet) \) represent a Zech’s logarithm transformation, then
\[
(1 + \beta^i)^2 = 1 + \beta^{2i}
\]
Proof: \( (1 + \beta^i)^2 = 1 + 2 \beta^i + \beta^{2i} = 1 + \beta^i + \beta^i + \beta^{2i} = 1 + \beta^{2i} \)

3.1.4 Zech’s Logarithm

The lists of field elements as polynomials, vectors, and their corresponding representations as powers of the generator \( \beta = x \), are given in Table 3.1 and Table 3.2. Using this we can easily build the Zech’s log table which can determine \( z(i) \) for each \( i \), \( 0 \leq i \leq p^m - 2 \).

The following is an example to illustrate how to obtain Zech’s log table. Table 3.3 is Zech’s log table for GF(2^3) generated by the root \( \beta = x \) of the irreducible polynomial \( f(x) = x^3 + x + 1 \). The definition of \( \beta^\infty = 0 \) makes Zech’s logarithm well-defined.
Table 3.3 Zech’s log table for GF(2³), defined by \( f(x) = x^3 + x + 1 \)

<table>
<thead>
<tr>
<th>( i )</th>
<th>( \beta^i )</th>
<th>( \beta^i )</th>
<th>( \beta^i + 1 )</th>
<th>( z(i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \infty )</td>
<td>( \beta^\infty ) (0)</td>
<td>000</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>( \beta^0 ) (1)</td>
<td>100</td>
<td>000</td>
<td>( \infty )</td>
</tr>
<tr>
<td>1</td>
<td>( \beta^1 )</td>
<td>010</td>
<td>110</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>( \beta^2 )</td>
<td>001</td>
<td>101</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>( \beta^3 = \beta + 1 )</td>
<td>110</td>
<td>010</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>( \beta^4 = \beta^2 + \beta )</td>
<td>011</td>
<td>111</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>( \beta^5 = \beta^4 + \beta + 1 )</td>
<td>111</td>
<td>011</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>( \beta^6 = \beta^3 + 1 )</td>
<td>101</td>
<td>001</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>( \beta^7 = 1 = \beta^0 )</td>
<td>100</td>
<td>000</td>
<td>0</td>
</tr>
</tbody>
</table>

Note that the product of zero and any number is still zero. That is,

\[ \beta^\infty \cdot \beta^k = \beta^{\infty + k} = \beta^\infty = 0, \] where \( k \) is random.

Alternatively, the Zech’s log table can often be constructed more rapidly in an ad-hoc manner, which is illustrated as follows. GF(2⁴) is taken in Table 3.4 as an example, where GF(2⁴) is defined by the \( f(x) = x^4 + x + 1 \). \( \beta \) is a root of \( x^4 + x + 1 \) and since + and – are equivalent in \( Z_2 \), \( \beta^4 + \beta + 1 = 0 \) and

\[ 1 + \beta = \beta^4 \quad (1) \]

This equation also implies \( 1 + \beta^4 = \beta \), providing table entries \( (i,z(i)) = (1,4) \) and \( (4,1) \).

Squaring both sides of equation (1) and using Lemma 3 (with \( p=2 \)), the relation is

\[ 1 + \beta^2 = \beta^8 \quad (2) \]

which yields entries \( (2,8) \) and \( (8,2) \). Now squaring both sides of (2) returns to relation (1) with nothing new. However, multiplying through (1) by \( \beta^{-1} \) gives

\[ \beta^{14} + 1 = \beta^3 \quad (3) \]
from which the entries (14,3) and (3,14) are derived. Squaring both sides of (3) provides entries (13,6) and (6,13). In a similar manner, multiplying by the inverse provides entries (11,12), (12,11), (7,9), (9,7), (5,10) and (10,5).

Table 3.4 Zech’s log table for GF(2^4), defined by \( f(x) = x^4 + x + 1 \)

<table>
<thead>
<tr>
<th>( i )</th>
<th>( z(i) )</th>
<th>( i )</th>
<th>( z(i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \infty )</td>
<td>0</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>( \infty )</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>14</td>
<td>3</td>
</tr>
</tbody>
</table>

The Zech’s log table for GF(2^6) which is generated by \( f(x) = x^6 + x + 1 \) is in Appendix A.

### 3.1.5 Calculation Based on Zech’s Logarithm

Once the Zech’s log table is established, the index \( i \) and the Zech’s logarithm \( z(i) \) columns of the Zech’s log table are stored in a read-only-memory (ROM) look-up table. Addition in GF(\( p^m \)) is then performed using the following scheme:

1. Combine all terms that have the same exponent using modular addition of the exponents (i.e., GF(p) addition)
2. Arrange the resulting expression $\beta^n + \beta^6 + \cdots + \beta^z$ in order of increasing exponents.

3. Combine the first two terms and take the biggest common item out, using the Zech’s log table to look for the summation of the rest of them, then multiply these two terms.

4. Repeat step 3 until only one term is left.

Therefore, the summation can now be performed as a series of add-one operations and Galois field multiplications.

With this table, elements of $F=GF(2^4)$ can be represented as powers of the generator $\beta=x$. Hence, both multiplication and addition are easily performed. For multiplication, simply add exponents and reduce modulo $q-1=15$. For example,

$$\beta^4 \beta^{13} = \beta^{17 \pmod{15}} = \beta^2$$

As an example of addition, to determine $\beta + \beta^3 + \beta^{11} + \beta^2 + \beta$ as a power of $\beta$, it is computed as follows:

$$\beta + \beta^3 + \beta^{11} + \beta^2 + \beta$$
$$= (\beta + \beta) + \beta^2 + \beta^3 + \beta^{11}$$
$$= \beta^2 + \beta^3 + \beta^{11} = \beta^2 (1 + \beta) + \beta^{11}$$
$$= \beta^2 \cdot \beta^4 + \beta^{11}$$
$$= \beta^6 + \beta^{11} = \beta^6 (1 + \beta^5)$$
$$= \beta^6 \cdot \beta^{10} = \beta^{16 \pmod{15}} = \beta$$

by using the Zech’s log table to look up $1 + \beta = \beta^4$, $1 + \beta^5 = \beta^{10}$. 
3.2 Permuted State Transition Table

In circular trellis-coded modulation (CTCM) with permuted state structure, the important requirement is building a permuted state transition table which guarantees the state constraint -- the starting state and ending state of a trellis path must be the same. Any input data block is uniquely mapped to a circular trellis path under this state transition table. In order to achieve the circular trellis path, the state transition table has to be properly designed. Design of the permuted state transition table involves the usage of Zech’s log table which is discussed in last section.

3.2.1 A Good Family and State Permutation

Before the building of the permuted state transition table, a good family needs to be obtained. The good family consists of good family members. Based on the derivation of Zech’s Logarithm, the good family can be constructed by state permutation.

One base member of a good family is called “natural (1, S-1)-type permutation” which presents two cycles. One cycle has only one element, and the other cycle has S-1 elements. Take (1)(2 3 4 … S-1) as an example of (1, S-1)-type permutation to explain. In (1)(2 3 4 … S-1), state 1 transits to state 1 itself in the first cycle. In the second cycle, state 2 transits to state 3, state 3 transits to state 4, and so on until state S-2 transits to state S-1 and the last state S-1 transits to state 2, which is back to the beginning of the second cycle. The first cycle is also called 1-self cycle. Therefore, the natural (1, S-1)-type permutation has the form of (1)(2 3 4 … S-1).
In a good family, the base family member is \((0)(\beta^1, \beta^2, \beta^3, ..., \beta^{S-2}, \beta^{S-1})\). It also can be denoted as state transition algebraically \((\infty)(1 2 3 4 ... S-2 S-1)\) (because \(\beta^\infty = 0\)), which is known as \(\sigma_\infty\).

The rest of the members in the good family can be easily achieved by adding \(\beta^i\) to the base member \(\sigma_\infty\), for \(1 \leq i \leq S - 1\). For a good member \(\sigma_i\), the state transition can be expressed as

\[
\sigma_i = (0 + \beta^i)(\beta^1 + \beta^i, \beta^2 + \beta^i, \beta^3 + \beta^i, ..., \beta^{S-2} + \beta^i, \beta^{S-1} + \beta^i)
\]

Formula 3.1 can be easily calculated by using Zech’s log table. Take \(i = 1\) as an example to illustrate the process.

\[
\sigma_1 = (0 + \beta)(\beta^1 + \beta, \beta^2 + \beta, \beta^3 + \beta, ..., \beta^{S-2} + \beta, \beta^{S-1} + \beta) = (\beta)(0, \beta \beta^{z(1)}, \beta \beta^{z(2)}, ..., \beta \beta^{z(S-3)}, \beta \beta^{z(S-2)})
\]

algebraically \(\Leftrightarrow\) \((1)(\infty, z(1) + 1, z(2) + 1, ..., z(S - 3) + 1, z(S - 2) + 1)\)

\(\Leftrightarrow\) \((1)(z(1) + 1, z(2) + 1, z(3) + 1, ..., z(S - 3) + 1, z(S - 2) + 1, \infty)\)

Therefore, \(\sigma_i\) can be written as

\[
\sigma_i = (i)(z(1) + i, z(2) + i, z(3) + i, ..., z(S - 2) + i, \infty)
\]

Table 3.5 is an example of a good family for 16 states, which is obtained by Formula 3.2 (State 0 = State 15 for consistence):
Table 3.5  A good family as $\sigma_i$ style for sixteen states

$$
\sigma_{\infty} = (\infty)(1,2,3,4,5,6,7,8,9,10,11,12,13,14,0)
\sigma_1 = (1)(5,9,0,2,11,14,10,3,8,6,13,12,7,4,\infty)
\sigma_2 = (2)(6,10,1,3,12,0,11,4,9,7,14,13,8,5,\infty)
\sigma_3 = (3)(7,11,2,4,13,1,12,5,10,8,0,14,9,6,\infty)
\sigma_4 = (4)(8,12,3,5,14,2,13,6,11,9,1,0,10,7,\infty)
\sigma_5 = (5)(9,13,4,6,0,3,14,7,12,10,2,1,11,8,\infty)
\sigma_6 = (6)(10,14,5,7,1,4,0,8,13,11,3,2,12,9,\infty)
\sigma_7 = (7)(11,0,6,8,2,5,1,9,14,12,4,3,13,10,\infty)
\sigma_8 = (8)(12,1,7,9,3,6,2,10,0,13,5,4,14,11,\infty)
\sigma_9 = (9)(13,2,8,10,4,7,3,11,14,6,5,0,12,\infty)
\sigma_{10} = (10)(14,3,9,11,5,8,4,12,2,0,7,6,1,13,\infty)
\sigma_{11} = (11)(0,4,10,12,6,9,5,13,3,1,8,7,2,14,\infty)
\sigma_{12} = (12)(1,5,11,13,7,10,6,14,4,2,9,8,3,0,\infty)
\sigma_{13} = (13)(2,6,12,14,8,11,7,0,5,3,10,9,4,1,\infty)
\sigma_{14} = (14)(3,7,13,0,9,12,8,1,6,4,11,10,5,2,\infty)
\sigma_0 = (0)(4,8,14,1,10,13,9,2,7,5,12,11,6,3,\infty)
$$

Additionally, there is another way to describe the good family according to the definition of state permutation which is shown in Table 3.6.
Table 3.6  A good family as state permutation style for sixteen states

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>∞</td>
<td>6</td>
<td>10</td>
<td>1</td>
<td>3</td>
<td>12</td>
<td>0</td>
<td>11</td>
<td>4</td>
<td>9</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td>∞</td>
<td>7</td>
<td>11</td>
<td>2</td>
<td>4</td>
<td>13</td>
<td>1</td>
<td>12</td>
<td>5</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>10</td>
<td>7</td>
<td>∞</td>
<td>8</td>
<td>12</td>
<td>3</td>
<td>5</td>
<td>14</td>
<td>2</td>
<td>13</td>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>11</td>
<td>8</td>
<td>∞</td>
<td>9</td>
<td>13</td>
<td>4</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>11</td>
<td>3</td>
<td>2</td>
<td>12</td>
<td>9</td>
<td>∞</td>
<td>10</td>
<td>14</td>
<td>5</td>
<td>7</td>
<td>1</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>14</td>
<td>12</td>
<td>4</td>
<td>3</td>
<td>13</td>
<td>10</td>
<td>∞</td>
<td>11</td>
<td>0</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>10</td>
<td>0</td>
<td>13</td>
<td>5</td>
<td>4</td>
<td>14</td>
<td>11</td>
<td>∞</td>
<td>12</td>
<td>1</td>
<td>7</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>3</td>
<td>11</td>
<td>1</td>
<td>14</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>12</td>
<td>∞</td>
<td>13</td>
<td>2</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>8</td>
<td>4</td>
<td>12</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>13</td>
<td>∞</td>
<td>14</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>6</td>
<td>9</td>
<td>5</td>
<td>13</td>
<td>3</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>14</td>
<td>∞</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>13</td>
<td>7</td>
<td>10</td>
<td>6</td>
<td>14</td>
<td>4</td>
<td>2</td>
<td>9</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>∞</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>8</td>
<td>11</td>
<td>7</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>10</td>
<td>9</td>
<td>4</td>
<td>1</td>
<td>∞</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>7</td>
<td>13</td>
<td>0</td>
<td>9</td>
<td>12</td>
<td>8</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>11</td>
<td>10</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>1</td>
<td>10</td>
<td>13</td>
<td>9</td>
<td>2</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>∞</td>
<td>5</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>11</td>
<td>14</td>
<td>10</td>
<td>3</td>
<td>8</td>
<td>6</td>
<td>13</td>
<td>12</td>
</tr>
</tbody>
</table>

| σ₁ | σ₁₂ | σ₁₃ | σ₁₄ | σ₀ | σ₁ | σ₂ | σ₃ | σ₄ | σ₅ | σ₆ | σ₇ | σ₈ | σ₉ | σ₁₀ | σ₁₁ |

The underlined numbers in above table are “i”s in each column of $\sigma_i$, which show the self-cycled numbers.

Once a good family is constructed, the existence and uniqueness of a good family which satisfies the state constraint needs to be proven. First, a theorem which proves that for the given family member $\sigma_\infty = (\infty) (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)$ is discussed.

The rest of the family members can be generated by this particular member via state permutation.

**Theorem 1**  To construct a good family for a given family member $\sigma_\infty = (\infty) (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)$, the rest of the good family members can be generated as $\sigma_i = \eta_i \sigma_\infty \eta_i^{-1}$, where $\eta_i(\gamma) = z(\gamma) + i$, by picking S-1 permutation $\eta_1, \eta_2, \ldots, \eta_{S-1}$. 
**Proof** Let \( \psi(j) = j + 1 \) and \( \eta_i = \psi^{i-1}\eta_1 \), such that

\[
\eta_i(\gamma) = \psi^{i-1}\eta_1(\gamma) = \psi^{i-1}(z(\gamma) + 1) = z(\gamma) + i
\]

Therefore, a good family contains \( \sigma_\infty, \sigma_1 = \eta_1\sigma_\infty\eta_1^{-1} \), and

\[
\sigma_i = \psi^{i-1}\sigma_1\psi^{-(i-1)} = \psi^{i-1}\eta_1\sigma_\infty\eta_1^{-1}\psi^{-(i-1)} = (\psi^{i-1}\eta_1)\sigma_\infty(\psi^{i-1}\eta_1)^{-1} = \eta_i\sigma_\infty\eta_i^{-1}
\]

Another theorem which proves the properties of existence and uniqueness by using the result of Theorem 1 will be discussed in next section – initial states.

### 3.2.2 State Table

Choosing four members from the good family forms a state table. The state table indicates what the next state is for any state driven by all possible transmission signals. For sixteen states, the state table is shown in Table 3.7. The four good family members are fixed, and they are \( \{\sigma_\infty, \sigma_{12}, \sigma_{14}, \sigma_5\} \). The corresponding trellis diagram is also explored in Figure 3.1.
Table 3.7 State transition table for 16 states

<table>
<thead>
<tr>
<th>Current State is</th>
<th>Next State is</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 transmitted</td>
</tr>
<tr>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\sigma_\infty$</td>
<td>$\sigma_{12}$</td>
</tr>
</tbody>
</table>
Figure 3.1 Trellis diagram of transmission-state set \( \{ \sigma_x, \sigma_{12}, \sigma_{14}, \sigma_3 \} \)

The next section includes an example which shows how to use the state table to determine the permuted state structure. Permutated state transition tables for 64 states are attached in Appendix B.

### 3.3 Initial States

#### 3.3.1 Theorem and Proof of Calculation of Initial State

As talked in the previous chapter, a permutated state transition table can be built which provides a circular Trellis path in tail-biting encoder. After get the circular trellis
path, the next step is to find an initial state, which will let the ending state be the same as the calculated initial state (aka, the starting state) for a given information sequence. For any information sequence, there exists one and only one initial state which achieves the same starting state and the ending state in the trellis path. Theorem 2 proves the properties of existence and uniqueness.

**Theorem 2** For any \((i_1, i_2, ..., i_B) \in (t^*)^B, t^* = \{x, 0, 1, ..., 14\}, B \neq 0 \text{(mod} t\text{)},\) there exists one and only one \(\alpha \in F\), such that

\[
\sigma_{i_B} \sigma_{i_{B-1}} \cdots \sigma_{i_2} \sigma_{i_1}(\alpha) = \alpha
\]

**Proof** Let \(B = 1\), then

\[
\sigma_i(\alpha) = \beta(\alpha - \beta^i) + \beta^i = \beta \alpha + \beta^i(1 - \beta)
\]

Assume it is true for \(B - 1\), then

\[
\begin{align*}
\sigma_{i_B} (\sigma_{i_{B-1}} \cdots \sigma_{i_1}(\alpha)) &= \beta[\sigma_{i_{B-1}} \cdots \sigma_{i_1}(\alpha) - \beta^{i_B}] + \beta^{i_B} \\
&= \beta[(\beta^{B-1} \alpha + (1 - \beta) \sum_{j=1}^{B-1} \beta^j + B - j) - \beta^{i_B}] + \beta^{i_B} \\
&= \beta^B \alpha + (1 - \beta) \sum_{j=1}^{B} \beta^{i_j + B - j}
\end{align*}
\]

Therefore,

\[
\alpha = (1 - \beta^B)^{-1} (1 - \beta) \sum_{j=1}^{B} \beta^{i_j + B - j} = \frac{z(1)}{z(B)} \sum_{j=1}^{B} \beta^{i_j + B - j}
\]

The Equation 3.2 shows the way to calculate the initial state, which guarantees the same starting state and the ending state through the trellis encoder for any given information sequence.
3.3.2 Mapping of Information Sequence for the Calculation of Initial States

In the state table mentioned in the previous section, there are four columns which only have the data of 1, 2, 3, 4 transmitted. However, there are all states (from $\infty$, 0 to $S-2$) in the Formula 3.2. Therefore, state mapping has to be done before the initial state can be calculated. It is an important step in calculation of the initial states.

Mapping the information sequence uses the state table. The basic steps are summarized as follows.

Given any information sequence $(i_1, i_2, ..., i_B) \in (t^*)^B$, $t^* = \{1,2,3,4\}$, $B \neq 0 \text{(mod} t\text{)}$, where $B$ is the block size:

1. Check the state table which has been used; find out which $\sigma_i$ is used for each column where 1,2,3,4 data are transmitted.
2. Substitute the subscribe number of $\sigma_i$ for the corresponding data of the information sequence.
3. Take the substitution sequence as a new sequence; put it into the Formula 3.2 to calculate the initial state which is for the original information sequence.
4. The calculated initial state should satisfy the requirement that the ending state is the same as the initial state, where the ending state is obtained after the initial state goes through the original information sequence.

3.3.3 Example of Calculation of Initial State

An example of how to calculate the initial state is shown as follows. In this example, the first step is state mapping.
Based on previous examples in the previous section, consider a given arbitrary transmitting sequence with block size 24,

\[ \{2, 1, 3, 1, 4, 1, 1, 1, 3, 4, 4, 3, 2, 4, 1, 2, 3, 4, 1, 3, 3, 1, 4, 3\} \]

The data of the sequence only can be 1, 2, 3, 4, since there are only two binary digits.

First, check the state table which is used in this project (Table 3.7).

It is found out that when data 1 is transmitted, \( \sigma_1 \) is being used; when 2 is transmitted, \( \sigma_{12} \) is being used; when 3 is transmitted, \( \sigma_{14} \) is being used; and when 4 is transmitted, \( \sigma_5 \) is being used. Therefore, the original sequence is mapped to

\[ (12, \infty, 14, \infty, 5, \infty, \infty, 14, 5, 5, 14, 12, 5, \infty, 12, 14, 5, \infty, 14, 14, 5, 14) \]

which is the corresponding \((i_1, i_2, \ldots, i_B)\), and \(B\) is 24.

The next step is to put them into the Equation 3.2, and resulting in

\[
\alpha = \beta^z(1) \cdot \beta^z(B) \cdot \sum_{j=1}^{24} \beta^{i_j+24-j} \\
= \beta^z(1) \cdot \beta^z(24) \cdot (\beta^{12+24-1} + \beta^{14+24-3} + \beta^{24-5} + \beta^{14+24-7} + \beta^{12+24-9} + \beta^{14+24-11} + \beta^{24-13} + \beta^{12+24-15} + \beta^{14+24-17} + \beta^{24-19} + \beta^{14+24-21} + \beta^{24-23} + \beta^{14+24-25}) \\
= \beta^z(1) \cdot (\beta^{35 \text{mod15}} + 0 + \beta^{35 \text{mod15}} + 0 + \beta^{24 \text{mod15}} + 0 + 0) \\
+ \beta^{29 \text{mod15}} + \beta^{19 \text{mod15}} + \beta^{18 \text{mod15}} + \beta^{16 \text{mod15}} + \beta^{15 \text{mod15}} + 0 + \beta^{20 \text{mod15}} \\
+ \beta^{21 \text{mod15}} + \beta^{11} + 0 + \beta^{18 \text{mod15}} + \beta^{17 \text{mod15}} + 0 + \beta^{6} + \beta^{14}) \\
= \beta^z \cdot (\beta^5 + \beta^5 + \beta^2 + \beta^4 + \beta^3 + \beta^1 + \beta^8 + \beta^6 + \beta^1 + \beta^3 + \beta^2 + \beta^6 + \beta^{14}) \\
= \beta^{3} \cdot (\beta^9 + \beta^4 + \beta^8 + 1 + \beta^5 + \beta^2) = \beta^{3} \cdot \beta^{12} \\
= \beta^9
\]

Therefore, state 9 is the initial state for this information sequence. Check if the ending state through the Trellis encoder is the same as the initial state 9 for this information sequence given above. The whole state transition path is:
The initial state is determined by the information sequence itself and can be any state in the trellis.

In order to make hardware for fast execution, the input information sequence has to be implemented parallel. That is, for information sequence with large block size, first separate the sequence into four sections, then do summations parallel while calculating the initial state. In the end, add these four summation results together to calculate the initial state for original sequence. This would increase the speed by a factor of four.

The Matlab program and C program to calculate the initial state are attached in Appendix D. Due to the property of programming, state 1 through state 16 are used in computer program instead of state $\infty$, and 0 through state 14 in the paper. Table 3.8 shows the minor state mapping from the paper to the computer program which let state $\infty \equiv 1$, $0 \equiv 16$, $others \equiv others + 1$. The state table is changed as well. State 10 is the initial state for the above example instead of state 9.
Table 3.8  State mapping from the paper to the computer program

<table>
<thead>
<tr>
<th>States in paper</th>
<th>States in program</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\infty$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>16</td>
</tr>
</tbody>
</table>
CHAPTER 4 PERFORMANCE

In this chapter, simulation results are given to explore the performance of the Circular Trellis-coded Modulation system. Error probability will be explored in the performance analysis. Additionally, the comparisons among different trellis depths, different dimensions, and different block sizes will be presented. The error probabilities will also be compared to the performance of BPSK.

4.1 Comparison of Bit Error Probabilities of CTCM Using Old and New Programs

Figure 4.1 Error probabilities comparison between old and new programs
These plots in Figure 4.1 show the difference between the simulation of old program and new programs of CTCM. The lines with “o” in these figures denote the simulation of new program of calculation of the initial states, and the lines with “+” represent the performance using the original program to calculate the initial states. There is no significant difference between the “o” lines and “+” lines.

4.2 Comparison of Bit Error Probabilities of CTCM with Different Block Sizes

The bit error probability in simulation is the number of different bits between the decoded information sequence and the information sequence input to the encoder over the total number of bits in the information sequence. Figure 4.2a shows the bit error probabilities for CTCM (12, 4, 2, B) where the block size B is respectively 24, 48 and 96. Figure 4.2b shows the bit error probabilities for CTCM (24, 4, 3, B) where the block size B is respectively 24, 48, and 96.
Figure 4.2  Error probabilities for B = 24, 48 and 96, and BPSK

The error probability improvement with increasing block size is easily seen from this plot. The performance improvement for increasing block size is in agreement with Shannon’s theorem (Golomb, 1994). One of the results of Shannon’s theorem states that the more source data that is encoded at once, the better the error performance will be. Therefore, for larger block sizes, the bit error probability improves. For example, the best curve for a block size of 96 in Figure 4.2 requires a bit energy-to-noise ratio of 1.25 dB to reach a value of $P_{be} = 10^{-5}$, whereas the curve for $B = 24$ reaches the same error probability for a bit energy-to-noise ratio of approximately 2.85 dB. In this simulation, the valid block size is less than 1024 and greater than 8. The performance of BPSK is also shown in the plot as a reference point.
4.3 Comparison of Bit Error Probabilities of CTCM with Different Dimensions

Figure 4.3a and 4.3b show bit error probability curves for block size $B = 24, 48$. Each plot compares performance using 12-dimension and 24-dimension signaling for 64-state trellis, along with the BPSK performance curve.

Since the 64-state trellis gives a wide range in the number of dimensions which can be utilized in constructing the transmission symbol table, the two extremes of $N = 12$ and $N = 24$ were chosen to be include in Figure 4.3. For each of the two plots, the 24-dimensional case offers slight improvement over the 12-dimensional case. For example,
in Figure 4.3 the curve for 24-dimension signaling requires a bit energy-to-noise ratio of 3.4dB to reach a value of \( P_{be} = 10^{-5} \), whereas the curve for 12-dimension reaches the same error probability for a bit energy-to-noise ratio of approximately 3.75 dB. Additionally, the performance improvement by larger block size also can be noticed by comparing plot a and plot b. The gain is apparent for 24-dimension signaling, where the \((B=48, S=64)\) case gives more than a decibel of gain over the \((B=24, S=64)\) case.

4.4 Comparison of Bit Error Probabilities of CTCM with Different Trellis Depths

Figure 4.4 Error probabilities for \( S = 16, 64 \) and BPSK
Figure 4.4 compares the error performance for 16-state and 64-state trellis. The performance of large state trellis is supposed to be better than low state trellis according to the large Euclidean distance the large state trellis can get. However, in this case, 64-state trellis is worse than 16-state trellis. There are some reasons to cause this problem. The first one is because of signal transmission symbol assignment table, which would give different distance distributions. Another possibility concerns simplex signaling where N has to be chosen properly. Future research will be included in the future work.
CHAPTER 5 CONCLUSIONS

The following is the conclusion of this thesis. A summary of the results obtained in this research will be included. Future research will be discussed.

5.1 Summary

This thesis develops and clarifies the theorem of calculation of the initial state for a circular trellis-coded modulation scheme utilizing a source alphabet size of $n=4$. Computer programs of Matlab® and C++ are included that perform the calculation of the initial state for any given information sequence. Error performance of both 16-state trellis and 64-state trellis are also presented.

Following the introduction of Chapter 1, Chapter 2 presents relevant background information, ranging from the concepts of digital communication system and error-correcting coding to the theory of trellis-coded modulation. Additionally, Chapter 2 introduces the idea of trellis termination, which is an important technique in trellis.

The design of the state table and the calculation of the initial states are explained in detail in Chapter 3. Zech’s logarithm is introduced first, which makes the calculation in Galois field much easier. Through the Zech’s logarithm, a good family can be constructed. Picking up four elements from the good family, the state table is built. The state table indicates what the next state is for any state driven by all possible transmission signals. Given any transmission sequence, in order to obtain a circular trellis path, one and only one initial state exists. A formula to calculate the initial state is given and proven in the end, and an example is shown, also.
Chapter 4 provides the performance of bit error probability of CTCM trellis. Comparisons are drawn for showing the affection of different CTCM parameters.

### 5.2 Future Research

First, figure out why the performance of high state trellis is not better than low state trellis.

Second, the simulation of the system would provide fairly accurate bit error probability curves at low signal-to-noise ratios. Therefore, the continuing of the simulation for more codewords and more parameter checking is necessary.

Finally, find a parallel implementation for hardware for fast execution.
REFERENCES


APPENDIX A

Zech’s Log Table for GF(2^6), Defined by \( f(x) = x^6 + x + 1 \)

<table>
<thead>
<tr>
<th>( i )</th>
<th>( z(i) )</th>
<th>( i )</th>
<th>( z(i) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \infty )</td>
<td>0</td>
<td>31</td>
<td>34</td>
</tr>
<tr>
<td>0</td>
<td>( \infty )</td>
<td>32</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>33</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>34</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>35</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>36</td>
<td>54</td>
</tr>
<tr>
<td>5</td>
<td>62</td>
<td>37</td>
<td>44</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>38</td>
<td>49</td>
</tr>
<tr>
<td>7</td>
<td>26</td>
<td>39</td>
<td>43</td>
</tr>
<tr>
<td>8</td>
<td>48</td>
<td>40</td>
<td>55</td>
</tr>
<tr>
<td>9</td>
<td>45</td>
<td>41</td>
<td>28</td>
</tr>
<tr>
<td>10</td>
<td>61</td>
<td>42</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>25</td>
<td>43</td>
<td>39</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>44</td>
<td>37</td>
</tr>
<tr>
<td>13</td>
<td>35</td>
<td>45</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>52</td>
<td>46</td>
<td>30</td>
</tr>
<tr>
<td>15</td>
<td>23</td>
<td>47</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>33</td>
<td>48</td>
<td>8</td>
</tr>
<tr>
<td>17</td>
<td>47</td>
<td>49</td>
<td>38</td>
</tr>
<tr>
<td>18</td>
<td>27</td>
<td>50</td>
<td>22</td>
</tr>
<tr>
<td>19</td>
<td>56</td>
<td>51</td>
<td>53</td>
</tr>
<tr>
<td>20</td>
<td>59</td>
<td>52</td>
<td>14</td>
</tr>
<tr>
<td>21</td>
<td>42</td>
<td>53</td>
<td>51</td>
</tr>
<tr>
<td>22</td>
<td>50</td>
<td>54</td>
<td>36</td>
</tr>
<tr>
<td>23</td>
<td>15</td>
<td>55</td>
<td>40</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
<td>56</td>
<td>19</td>
</tr>
<tr>
<td>25</td>
<td>11</td>
<td>57</td>
<td>58</td>
</tr>
<tr>
<td>26</td>
<td>7</td>
<td>58</td>
<td>57</td>
</tr>
<tr>
<td>27</td>
<td>18</td>
<td>59</td>
<td>20</td>
</tr>
<tr>
<td>28</td>
<td>41</td>
<td>60</td>
<td>29</td>
</tr>
<tr>
<td>29</td>
<td>60</td>
<td>61</td>
<td>10</td>
</tr>
<tr>
<td>30</td>
<td>46</td>
<td>62</td>
<td>5</td>
</tr>
</tbody>
</table>
APPENDIX B

Permutated State Transition Table for 64 States

<table>
<thead>
<tr>
<th>Current state is</th>
<th>Next state is</th>
<th>1 transmitted</th>
<th>2 transmitted</th>
<th>3 transmitted</th>
<th>4 transmitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\infty$</td>
<td>$\infty$</td>
<td>1</td>
<td>4</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>14</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>13</td>
<td>9</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>33</td>
<td>$\infty$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>25</td>
<td>10</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>0</td>
<td>16</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>2</td>
<td>36</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>27</td>
<td>28</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>49</td>
<td>3</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>46</td>
<td>5</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>62</td>
<td>30</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>26</td>
<td>52</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>3</td>
<td>49</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>36</td>
<td>2</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>53</td>
<td>29</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>24</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>34</td>
<td>39</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>48</td>
<td>56</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>28</td>
<td>27</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>57</td>
<td>37</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>60</td>
<td>51</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>43</td>
<td>31</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>51</td>
<td>60</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>24</td>
<td>16</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>25</td>
<td>5</td>
<td>46</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>12</td>
<td>54</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>27</td>
<td>8</td>
<td>19</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>28</td>
<td>19</td>
<td>8</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>29</td>
<td>42</td>
<td>15</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>30</td>
<td>61</td>
<td>11</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>31</td>
<td>47</td>
<td>22</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>35</td>
<td>45</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>33</td>
<td>4</td>
<td>1</td>
<td>$\infty$</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>34</td>
<td>17</td>
<td>50</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>35</td>
<td>32</td>
<td>38</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>36</td>
<td>14</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>37</td>
<td>55</td>
<td>20</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>38</td>
<td>45</td>
<td>35</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>39</td>
<td>50</td>
<td>17</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>40</td>
<td>44</td>
<td>58</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>41</td>
<td>56</td>
<td>48</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>42</td>
<td>29</td>
<td>53</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>43</td>
<td>22</td>
<td>47</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>44</td>
<td>40</td>
<td>59</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>45</td>
<td>38</td>
<td>32</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>46</td>
<td>10</td>
<td>25</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>47</td>
<td>31</td>
<td>43</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>48</td>
<td>18</td>
<td>41</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>49</td>
<td>9</td>
<td>13</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>50</td>
<td>39</td>
<td>34</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>51</td>
<td>23</td>
<td>21</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>52</td>
<td>54</td>
<td>12</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>53</td>
<td>15</td>
<td>42</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>54</td>
<td>52</td>
<td>26</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>55</td>
<td>37</td>
<td>57</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>56</td>
<td>41</td>
<td>18</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>57</td>
<td>20</td>
<td>55</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>58</td>
<td>59</td>
<td>40</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>59</td>
<td>58</td>
<td>44</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>60</td>
<td>21</td>
<td>23</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>61</td>
<td>30</td>
<td>62</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>62</td>
<td>11</td>
<td>61</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>0</td>
<td>6</td>
<td>24</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$\infty$</td>
<td>33</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>??</td>
<td>$\sigma_{\infty}$</td>
<td>$\sigma_{12}$</td>
<td>$\sigma_{14}$</td>
<td>$\sigma_{5}$</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX C
Matlab® Program of Calculation of Initial States

%function ini=init(n,S,data)
clear;
n=4;D=2;S=n^D;
if S>16
    clear;
    data=[2 1 3 1 4 1 1 1 3 4 4 3 2 4 1 2 3 4 1 3 1 4 3] % data only can be 1 through 4
    dd=length(data); % dd ~= n*15 = n*(S-1)
    [i zech]=zech_gen_1to16(log2(S)); % What I changed in zech_play_inf.m from zech_gen.m is adding z(inf)=0, z(0)=inf.
    st=st_gen(4,2)
    for k=1:4
        z(1,k)=find((1:S)'==st(:,k)); % z is 1, 13, 15, 6
    end
    z=z-1; % 1 <-> inf
    % 16 <-> 0
    % others <-> others -1
    % Calculating the results of data(j)+B-j, B is the length of data
    new=[];
    acc=[];
    for j=1:dd
        new=[new z(data(j))]; % Calculating the summation of items in acc matrix (using zech-log table)
        if z(data(j))~0
            cc=rem(z(data(j))+dd-j,S-1);
            if cc<0
                cc=cc+S-1;
            end
            acc=[acc cc];
        end
    end
    new;
    acc;
    % Then put the summation result as the last one in acc matrix
    num=length(acc);
    tempFlag = 1;
    for j=1:num-1
        if tempFlag==1
            if acc(j)==acc(j+1)
                diff=abs(acc(j+1)-acc(j));
            end
        end
    end
end
acc(j)=min(acc(j),acc(j+1));
acc(j+1)=zech(find(i==diff));
acc(j+1)=rem(acc(j)+acc(j+1),S-1);
else
  acc(j+1)=inf;
tempFlag=0;
end
else
  tempFlag=1;
end
end
end

acc;

k=rem(dd,S-1);  % k is 1 through 14
num=length(acc);
if acc(num)==inf
  ini=1;
else
  ini=zech(find(i==1))-zech(find(i==k))+acc(num)  % z(1)-z(k)+summation
  ini=rem(ini+S-1,S-1);   % 0,1,...,13,14   (15)
  if ini==0              % 16,1,...,13,14   (15)
    ini=S;
  else
    ini=ini+1;            % 16,2,...,14,15   (15)
  end
end
ini
APPENDIX D

C++ Program of Calculation of Initial States

// find initial state for "a" sequence
// Changed by Yiqi Chen

dd=Bmax;       // length of data
int *acc = new int(dd);
int count=0;
for (j=0; j<dd; j++) {
    if (z[a_data[j]]!=0) {
        int cc = (z[a_data[j]]+dd-j)%(S-1);
        if (cc<=0) {
            cc=cc+S-1;
        }
        acc[count]=cc;
        count++;
    }
}

int num=count;
for (j=0; j<num-1;) {
    if (acc[j]!=acc[j+1]) {
        diff=abs(acc[j+1]-acc[j]);
        if(acc[j]>acc[j+1]) {
            acc[j] = acc[j+1];
        }
        acc[j+1]=zech[diff];
        acc[j+1]=(acc[j]+acc[j+1])%(S-1);
        j++;
    }
    else {
        acc[j+1]=S;
        j=j+2;
    }
}

int k=dd%(S-1);    // k is 1 through 14
if (acc[num-1]==S) {
    a_initial_state=1;
}
else {
    a_initial_state=zech[0]-zech[k-1]+acc[num-1];
    if (a_initial_state==0) {

a_initial_state=S;
}

else {
    a_initial_state=(a_initial_state+1)%(S-1);
}

if (a_initial_state<0) {
    a_initial_state=a_initial_state+S-1;
} // end of changing for 'a' sequence